analyse

September 4, 2020

[1]:
```
import cdsapi

import numpy as np
import pandas as pd
import xarray as xr
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.rcParams['figure.figsize'] = (12,8)
```

[2]:
```
c = cdsapi.Client()
```

[3]:
```
c.retrieve(
    'reanalysis-era5-land',
    {
        'format': 'netcdf',
        'variable':[
            '2m_temperature','total_precipitation',␣
 ↪'volumetric_soil_water_layer_1',
        ],
        'year':'2019',
        'month':'12',
        'day':[
            '01', '02', '03',
            '04', '05', '06',
            '07', '08', '09',
            '10', '11', '12',
            '13', '14', '15',
            '16', '17', '18',
            '19', '20', '21',
            '22', '23', '24',
            '25', '26', '27',
            '28', '29', '30',
            '31',
        ],
    },
    'download.nc')
```

```
2020-09-02 20:16:23,437 INFO Welcome to the CDS
2020-09-02 20:16:23,440 INFO Sending request to
https://cds.climate.copernicus.eu/api/v2/resources/reanalysis-era5-land
2020-09-02 20:16:24,528 INFO Request is queued
2020-09-02 20:16:27,329 INFO Request is running
2020-09-02 20:17:40,754 INFO Request is completed
2020-09-02 20:17:40,755 INFO Downloading http://136.156.133.46/cache-compute-001
5/cache/data4/adaptor.mars.internal-1599057987.4776475-14712-20-d6bf919f-5d56-4f
87-9e9d-68f1b9034106.nc to download.nc (1.1G)
2020-09-02 20:22:14,747 INFO Download rate 4.2M/s
```

[3]:
```
Result(content_length=1205972896,content_type=application/x-netcdf,location=http
://136.156.133.46/cache-compute-0015/cache/data4/adaptor.mars.internal-159905798
7.4776475-14712-20-d6bf919f-5d56-4f87-9e9d-68f1b9034106.nc)
```

[4]:
```
ds = xr.open_dataset('download.nc')
```

[5]:
```
ds
```

[5]:
```
<xarray.Dataset>
Dimensions:    (latitude: 1801, longitude: 3600, time: 31)
Coordinates:
  * longitude  (longitude) float32 0.0 0.1 0.2 0.3 … 359.6 359.7 359.8 359.9
  * latitude   (latitude) float32 90.0 89.9 89.8 89.7 … -89.8 -89.9 -90.0
  * time       (time) datetime64[ns] 2019-12-01T12:00:00 … 2019-12-31T12:00:00
Data variables:
    t2m        (time, latitude, longitude) float32 …
    tp         (time, latitude, longitude) float32 …
    swvl1      (time, latitude, longitude) float32 …
Attributes:
    Conventions:  CF-1.6
    history:      2020-09-02 14:47:01 GMT by grib_to_netcdf-2.16.0: /opt/ecmw…
```

1 Exploring and Visualising Geospatial Data

1.1 Calculating Basic Statistics

1.1.1 Temperature of air at 2m above the surface

[6]:
```
ds.t2m
```

[6]:
```
<xarray.DataArray 't2m' (time: 31, latitude: 1801, longitude: 3600)>
[200991600 values with dtype=float32]
Coordinates:
  * longitude  (longitude) float32 0.0 0.1 0.2 0.3 … 359.6 359.7 359.8 359.9
  * latitude   (latitude) float32 90.0 89.9 89.8 89.7 … -89.8 -89.9 -90.0
  * time       (time) datetime64[ns] 2019-12-01T12:00:00 … 2019-12-31T12:00:00
```

```
Attributes:
    units:      K
    long_name:  2 metre temperature
```

[7]:
```
ds.t2m.min()
```

[7]:
```
<xarray.DataArray 't2m' ()>
array(221.28519, dtype=float32)
```

[8]:
```
ds.t2m.max()
```

[8]:
```
<xarray.DataArray 't2m' ()>
array(317.77365, dtype=float32)
```

[9]:
```
ds.t2m.mean()
```

[9]:
```
<xarray.DataArray 't2m' ()>
array(268.24347, dtype=float32)
```

[10]:
```
ds.t2m.median()
```

[10]:
```
<xarray.DataArray 't2m' ()>
array(264.078, dtype=float32)
```

[11]:
```
ds.t2m.std()
```

[11]:
```
<xarray.DataArray 't2m' ()>
array(21.81747, dtype=float32)
```

[12]:
```
ds.t2m.var()
```

[12]:
```
<xarray.DataArray 't2m' ()>
array(476.00198, dtype=float32)
```

1.1.2 Total Precipitation

[13]:
```
ds.tp
```

[13]:
```
<xarray.DataArray 'tp' (time: 31, latitude: 1801, longitude: 3600)>
[200991600 values with dtype=float32]
Coordinates:
  * longitude  (longitude) float32 0.0 0.1 0.2 0.3 … 359.6 359.7 359.8 359.9
  * latitude   (latitude) float32 90.0 89.9 89.8 89.7 … -89.8 -89.9 -90.0
  * time       (time) datetime64[ns] 2019-12-01T12:00:00 … 2019-12-31T12:00:00
Attributes:
    units:      m
    long_name:  Total precipitation
```

[14]:
```
ds.tp.min()
```

[14]:
```
<xarray.DataArray 'tp' ()>
array(7.450581e-09, dtype=float32)
```

[15]:
```
ds.tp.max()
```

[15]:
```
<xarray.DataArray 'tp' ()>
array(0.21889278, dtype=float32)
```

[16]:
```
ds.tp.mean()
```

[16]:
```
<xarray.DataArray 'tp' ()>
array(0.00069312, dtype=float32)
```

[17]:
```
ds.tp.median()
```

[17]:
```
<xarray.DataArray 'tp' ()>
array(1.6704202e-05, dtype=float32)
```

[18]:
```
ds.tp.std()
```

[18]:
```
<xarray.DataArray 'tp' ()>
array(0.0025754, dtype=float32)
```

[19]:
```
ds.tp.var()
```

[19]:
```
<xarray.DataArray 'tp' ()>
array(6.632711e-06, dtype=float32)
```

1.1.3 Volumetric soil water layer 1

[20]:
```
ds.swvl1
```

[20]:
```
<xarray.DataArray 'swvl1' (time: 31, latitude: 1801, longitude: 3600)>
[200991600 values with dtype=float32]
Coordinates:
  * longitude  (longitude) float32 0.0 0.1 0.2 0.3 … 359.6 359.7 359.8 359.9
  * latitude   (latitude) float32 90.0 89.9 89.8 89.7 … -89.8 -89.9 -90.0
  * time       (time) datetime64[ns] 2019-12-01T12:00:00 … 2019-12-31T12:00:00
Attributes:
    units:      m**3 m**-3
    long_name:  Volumetric soil water layer 1
```

[21]:
```
ds.swvl1.min()
```

[21]: <xarray.DataArray 'swvl1' ()>
array(0., dtype=float32)

[22]:
```
ds.swvl1.max()
```

[22]: <xarray.DataArray 'swvl1' ()>
array(0.76600647, dtype=float32)

[23]:
```
ds.swvl1.mean()
```

[23]: <xarray.DataArray 'swvl1' ()>
array(0.2649494, dtype=float32)

[24]:
```
ds.swvl1.median()
```

[24]: <xarray.DataArray 'swvl1' ()>
array(0.2716025, dtype=float32)

[25]:
```
ds.swvl1.std()
```

[25]: <xarray.DataArray 'swvl1' ()>
array(0.13019994, dtype=float32)

[26]:
```
ds.swvl1.var()
```

[26]: <xarray.DataArray 'swvl1' ()>
array(0.01695202, dtype=float32)

1.2 Checking for missing values

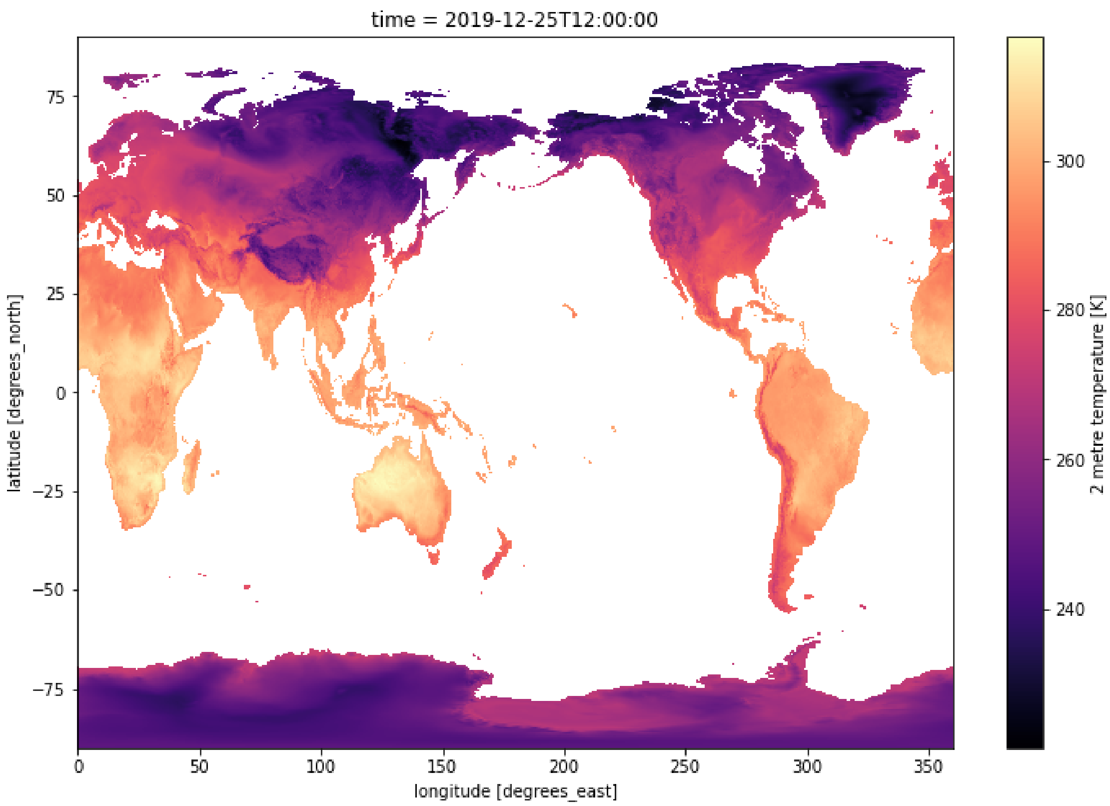
[27]:
```
ds.isnull().count()
```

[27]:
```
<xarray.Dataset>
Dimensions:  ()
Data variables:
    t2m      int32 200991600
    tp       int32 200991600
    swvl1    int32 200991600
```

1.3 Plotting data

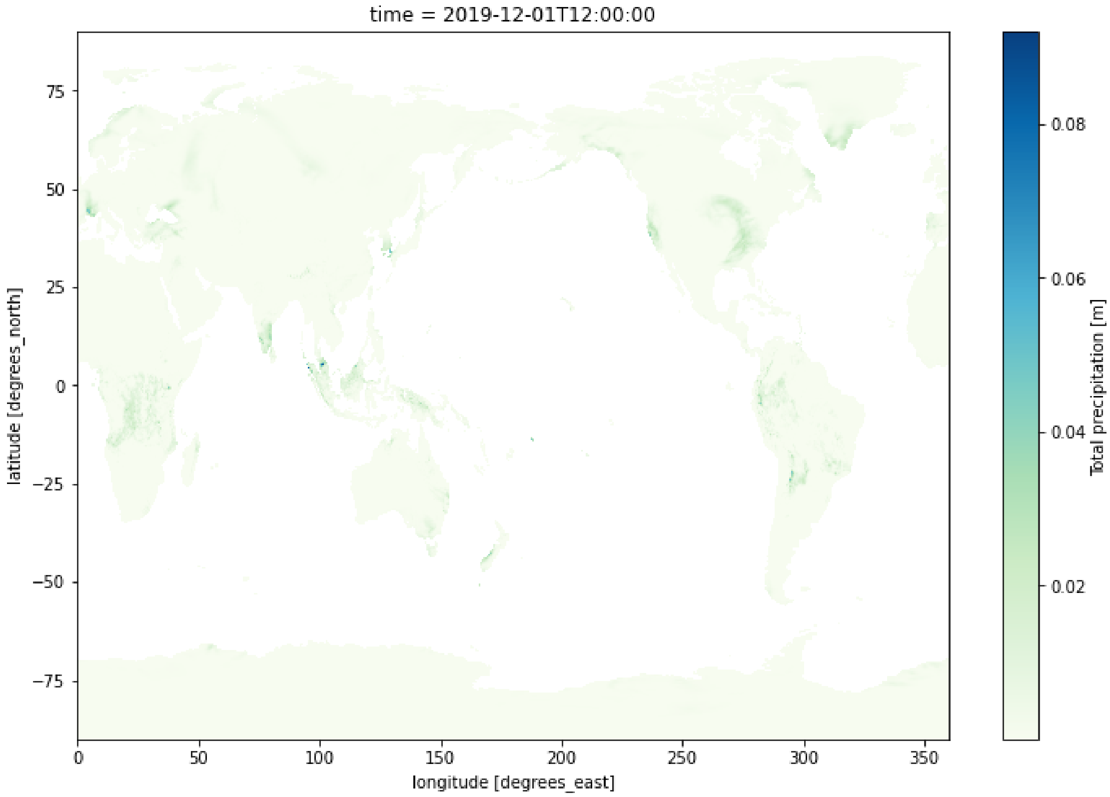
[28]:
```
ds.t2m.sel(time='2019-12-25').plot(cmap='magma')
```

[28]: <matplotlib.collections.QuadMesh at 0x15860690f10>



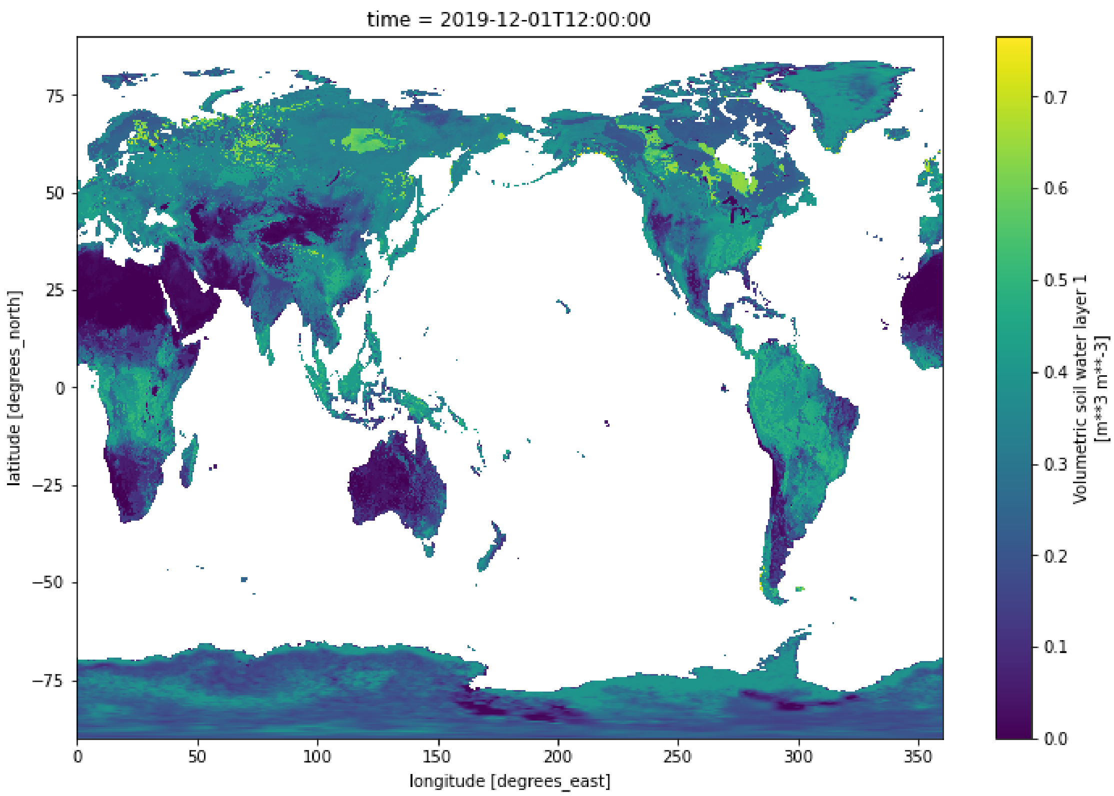
[29]:
```
ds.tp.sel(time='2019-12-01').plot(cmap='GnBu')
```

[29]: <matplotlib.collections.QuadMesh at 0x158607602b0>



[30]:
```
ds.swvl1.sel(time='2019-12-01').plot()
```

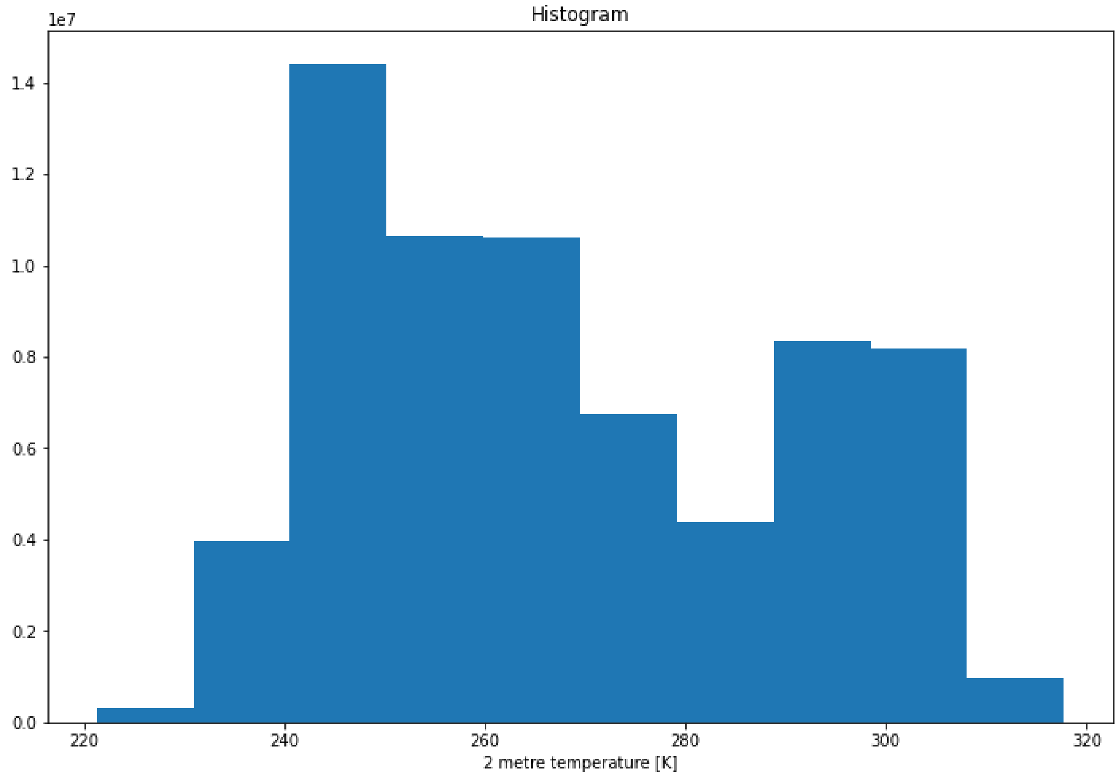
[30]: <matplotlib.collections.QuadMesh at 0x15860af4a90>



1.4 Visualising data distribution

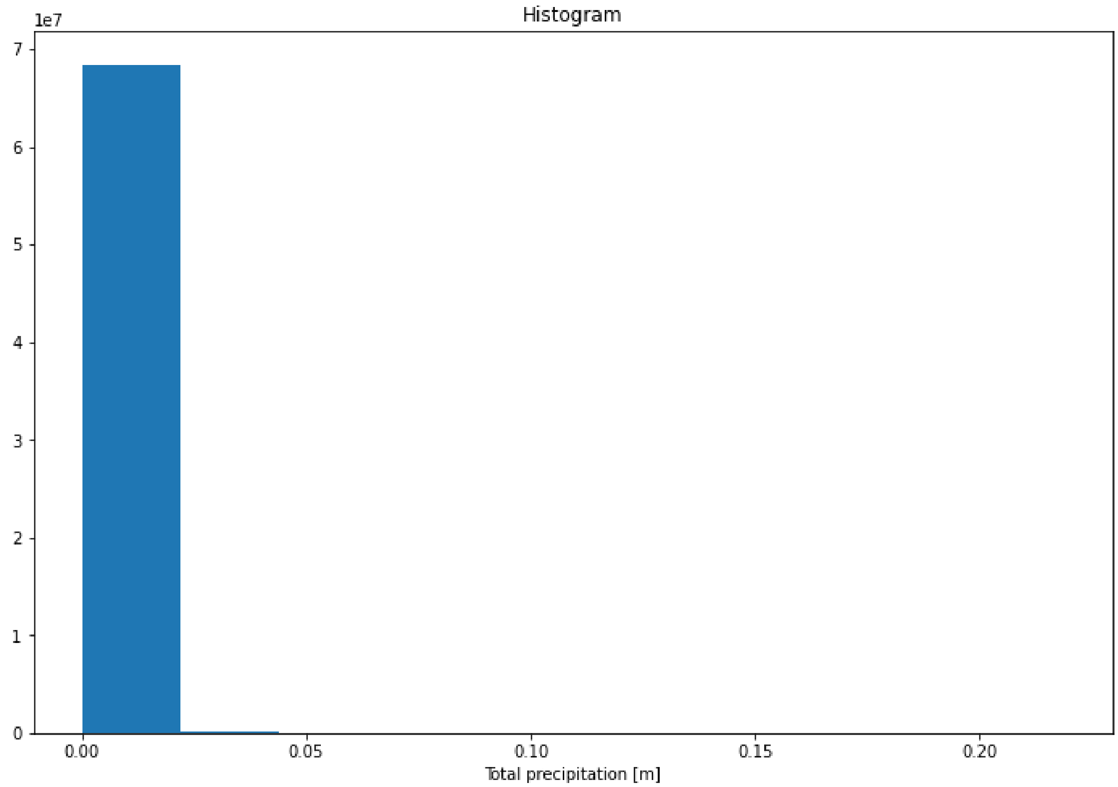
[31]:
```
ds.t2m.plot()
```

[31]:
```
(array([  321574.,  3954441., 14419857., 10647769., 10623027.,  6757909.,
         4377244.,  8346348.,  8174480.,   976104.]),
 array([221.28519, 230.93404, 240.58289, 250.23172, 259.88058, 269.52942,
        279.17825, 288.82712, 298.47595, 308.12482, 317.77365],
       dtype=float32),
 <BarContainer object of 10 artists>)
```



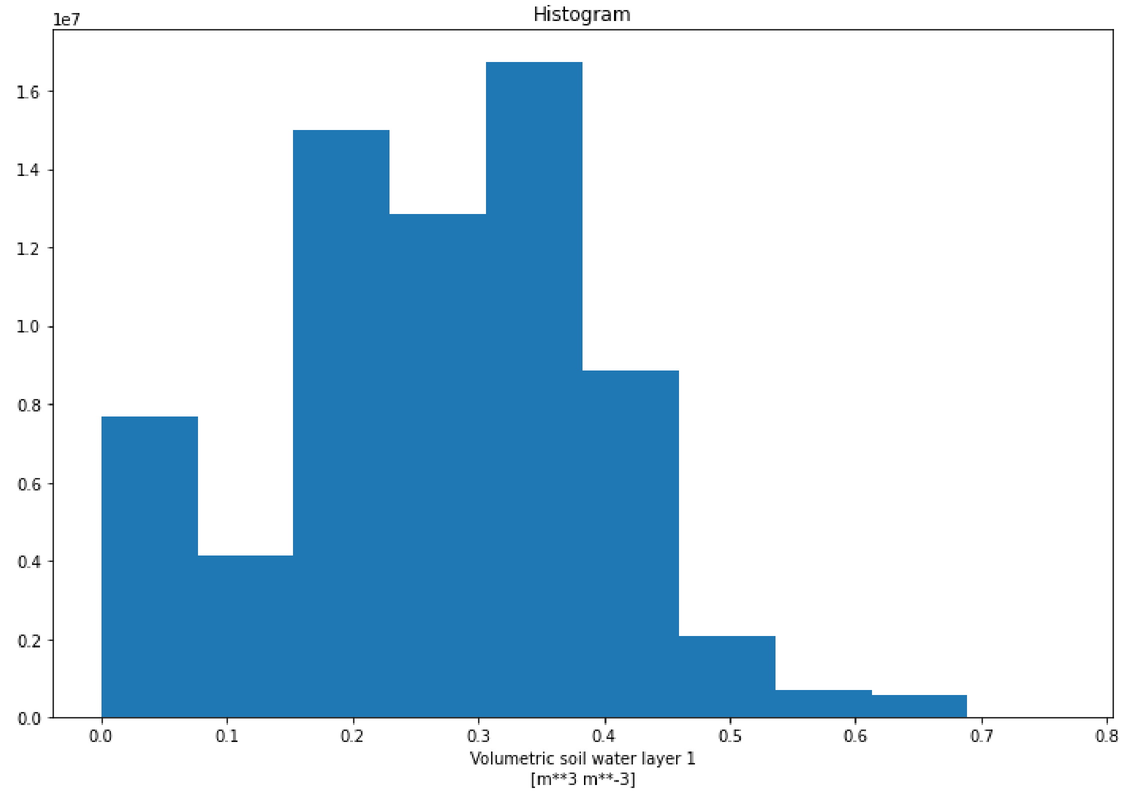
[32]:
```
ds.tp.plot()
```

[32]:
```
(array([6.8436371e+07, 1.4117100e+05, 1.5473000e+04, 3.8770000e+03,
        1.1820000e+03, 4.5600000e+02, 1.6200000e+02, 5.2000000e+01,
        7.0000000e+00, 2.0000000e+00]),
 array([7.4505806e-09, 2.1889284e-02, 4.3778561e-02, 6.5667838e-02,
        8.7557115e-02, 1.0944639e-01, 1.3133568e-01, 1.5322495e-01,
        1.7511423e-01, 1.9700350e-01, 2.1889278e-01], dtype=float32),
 <BarContainer object of 10 artists>)
```



[33]:
```
ds.swvl1.plot()
```

[33]:
```
(array([ 7662084.,  4145751., 14999080., 12867079., 16728351.,  8844565.,
         2057516.,   700088.,   566021.,    28218.]),
 array([0.        , 0.07660065, 0.1532013 , 0.22980194, 0.3064026 ,
        0.38300323, 0.45960388, 0.5362045 , 0.6128052 , 0.6894058 ,
        0.76600647], dtype=float32),
 <BarContainer object of 10 artists>)
```



2 Preprocessing Geospatial Data

2.1 Interpolation

[34]:
```
ds.t2m.resample(time='2D').interpolate('linear')
```

```
C:\Users\imdcl\Anaconda3\envs\cds\lib\site-packages\xarray\core\common.py:1123:
FutureWarning: 'base' in .resample() and in Grouper() is deprecated.
The new arguments that you should use are 'offset' or 'origin'.

>>> df.resample(freq="3s", base=2)

becomes:

>>> df.resample(freq="3s", offset="2s")

  grouper = pd.Grouper(
```

[34]:
```
<xarray.DataArray 't2m' (time: 16, latitude: 1801, longitude: 3600)>
array([[[          nan,           nan,           nan, …,           nan,
                   nan,           nan],
```

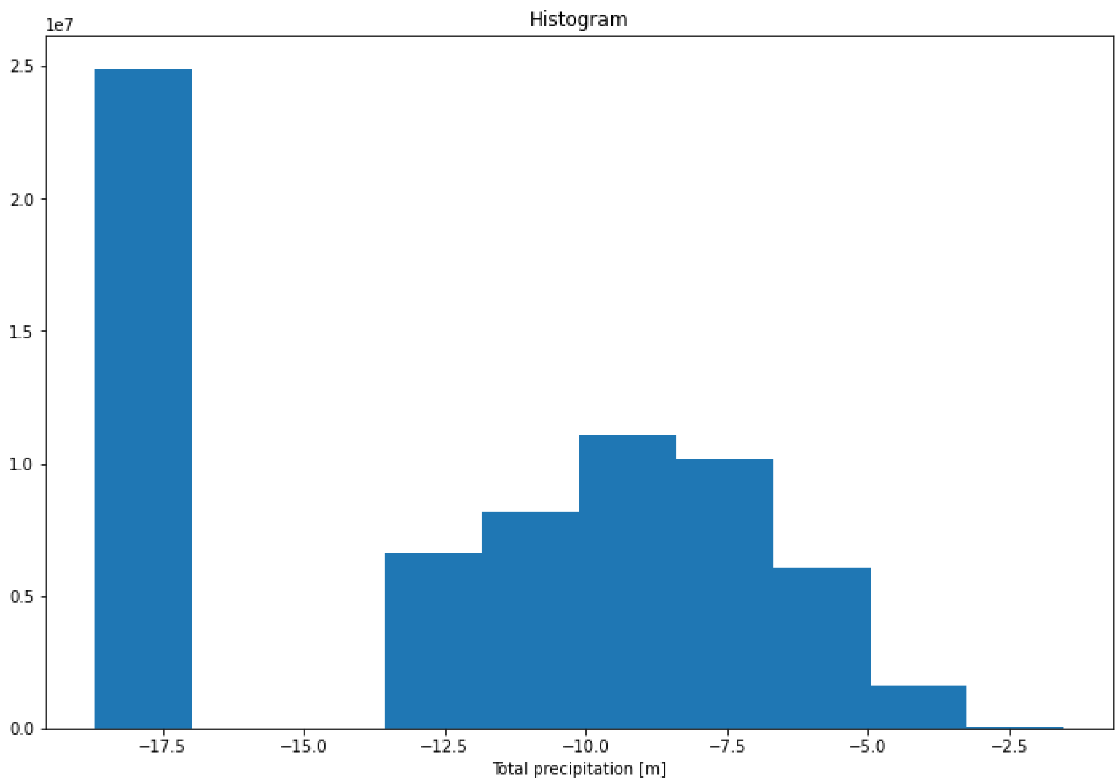
```
        [          nan,          nan,          nan, …,          nan,
                   nan,          nan],
        [          nan,          nan,          nan, …,          nan,
                   nan,          nan],
        …,
        [          nan,          nan,          nan, …,          nan,
                   nan,          nan],
        [          nan,          nan,          nan, …,          nan,
                   nan,          nan],
        [          nan,          nan,          nan, …,          nan,
                   nan,          nan]],

       [[          nan,          nan,          nan, …,          nan,
                   nan,          nan],
        [          nan,          nan,          nan, …,          nan,
                   nan,          nan],
        [          nan,          nan,          nan, …,          nan,
                   nan,          nan],
…
        [247.91584778, 247.91437531, 247.91437531, …, 247.91510773,
         247.91584778, 247.91584778],
        [247.89965057, 247.89965057, 247.89891052, …, 247.89965057,
         247.89965057, 247.89965057],
        [247.62726593, 247.62726593, 247.62726593, …, 247.62726593,
         247.62726593, 247.62726593]],

       [[          nan,          nan,          nan, …,          nan,
                   nan,          nan],
        [          nan,          nan,          nan, …,          nan,
                   nan,          nan],
        [          nan,          nan,          nan, …,          nan,
                   nan,          nan],
        …,
        [246.31685638, 246.31685638, 246.31685638, …, 246.31096649,
         246.31317902, 246.31611633],
        [246.29845428, 246.29845428, 246.29845428, …, 246.29255676,
         246.29403687, 246.29624176],
        [245.80374146, 245.80374146, 245.80374146, …, 245.80374146,
         245.80374146, 245.80374146]]])
Coordinates:
  * longitude  (longitude) float32 0.0 0.1 0.2 0.3 … 359.6 359.7 359.8 359.9
  * latitude   (latitude) float32 90.0 89.9 89.8 89.7 … -89.8 -89.9 -90.0
  * time       (time) datetime64[ns] 2019-12-01 2019-12-03 … 2019-12-31
Attributes:
    units:      K
    long_name:  2 metre temperature
```

2.2 Transformation

[35]:
```
ds.tp.data = np.log(ds.tp.data)
```

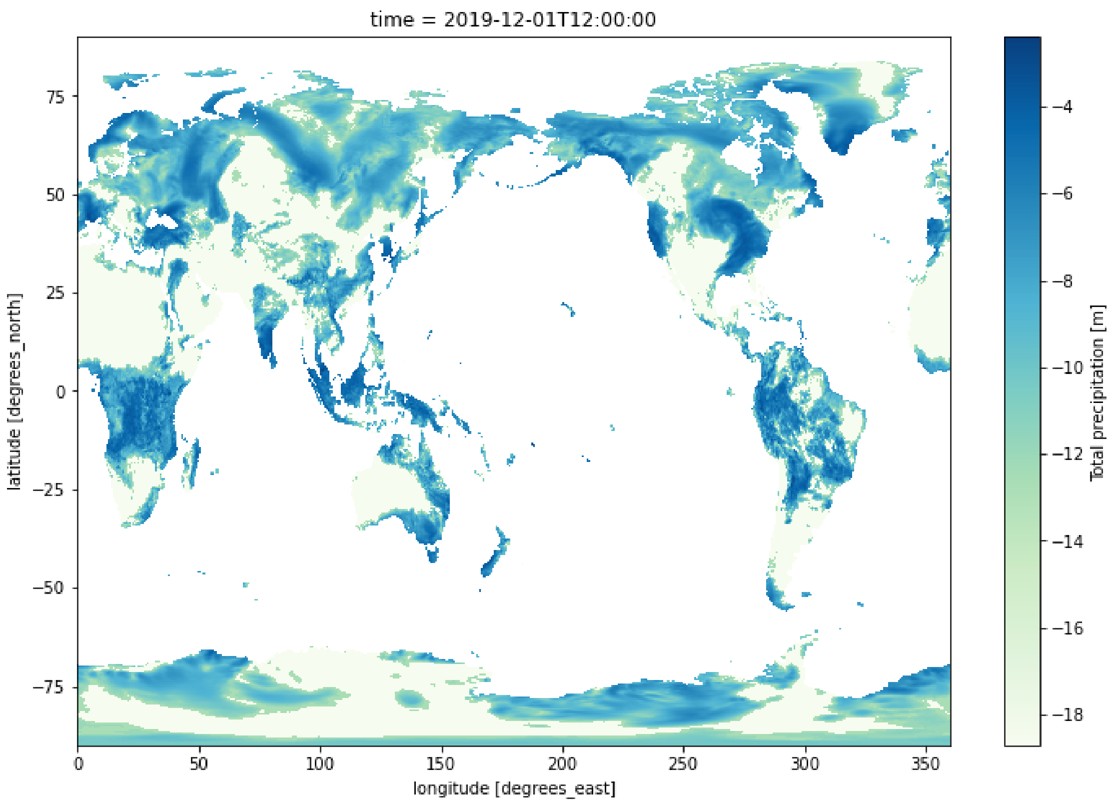
[36]:
```
ds.tp.plot()
```

[36]:
```
(array([24913495.,        0.,        0.,  6602140.,  8163657., 11066456.,
        10153525.,  6054693.,  1615693.,    29094.]),
 array([-18.714973 , -16.995394 , -15.275813 , -13.556233 , -11.836654 ,
        -10.117073 ,  -8.397493 ,  -6.677913 ,  -4.9583335,  -3.2387533,
         -1.5191733], dtype=float32),
 <BarContainer object of 10 artists>)
```



[37]:
```
ds.tp.sel(time='2019-12-01').plot(cmap='GnBu')
```

[37]: <matplotlib.collections.QuadMesh at 0x158627f15e0>



[38]:
```
ds.tp.min()
```

[38]:
```
<xarray.DataArray 'tp' ()>
array(-18.714973, dtype=float32)
```

[39]:
```
ds.tp.max()
```

[39]:
```
<xarray.DataArray 'tp' ()>
array(-1.5191733, dtype=float32)
```

[40]:
```
ds.tp.mean()
```

[40]:
```
<xarray.DataArray 'tp' ()>
array(-12.518761, dtype=float32)
```

[41]:
```
ds.tp.median()
```

[41]:
```
<xarray.DataArray 'tp' ()>
array(-10.99985, dtype=float32)
```

[42]:
```
ds.tp.std()
```

[42]: <xarray.DataArray 'tp' ()>
array(5.0175514, dtype=float32)

[43]:
```
ds.tp.var()
```

[43]: <xarray.DataArray 'tp' ()>
array(25.175823, dtype=float32)

3 Deep Learning for Geospatial Data

3.1 Questions

3.1.1 How you will split the data for training, validation and testing?

As I am working with geospatial data, I will split the data with respect to the regions as spliting the data randomly will result in a biased training which will result in an overfitted model.

3.1.2 Implementations for data loading, data transformation & inverse transformation

[44]:
```
# add code here
```

3.1.3 Obtaining and fine-tuning a pre-trained model

I will go throught the literature of deep learning for geospatial data to find out what architectures give the best results on our data.

3.1.4 Function descriptions and definitions for model training, testing and inference

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        epoch_time = time.time()
        print('Epoch {}/{}'.format(epoch, num_epochs-1))
        print('-'*10)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0
```

```
        for inputs, labels in dataloaders[phase]:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)

                if phase == 'train':
                    loss.backward()
                    optimizer.step()

            running_loss += loss.item()*inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

        if phase == 'train':
            scheduler.step()

        epoch_loss = running_loss/dataset_sizes[phase]
        epoch_acc = running_corrects.double()/dataset_sizes[phase]
        epoch_elapsed = time.time()-epoch_time
        print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epoch_acc))

        if phase == 'val' and epoch_acc>best_acc:
            best_acc = epoch_acc
            best_model_wts = copy.deepcopy(model.state_dict())

    print('Epoch time: {:.0f}m {:.0f}s'.format(epoch_elapsed//60, epoch_elapsed%60))
    print()

time_elapsed = time.time() - since
print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed//60, time_elapsed%60))
print('Best val Acc: {:4f}'.format(best_acc))

model.load_state_dict(best_model_wts)
return model
```

3.1.5 Your choice of activation function, loss function and error metrics

Again, I will go throught the literature to find out the best combination of evalution metrics.

3.1.6 Implementation techniques that help improve the efficient of model training and dataloading

Dropout is a good method which improves the efficiency of the model and avoids overfitting. Pruning is also proved to improved the performance of a model which allows the model to be run on mobile devices.

3.1.7 How will you avoid overfitting?

Using a good archichtecture with dropouts will help in avoid overfitting.

3.1.8 What do you use for visualizing model training and performance?

I will plot my predictions and compare them to ground truth.

3.1.9 What factors do you think might restrict the model from achieving a high accuracy?

Not spliting the data properly will result in an overfitted model which in turn will restrict the model to have high accuracy.